

[About Source Code Management](#)

[STARTING WITH GIT](#)

[GIT CONFIG](#)

[GIT HELP](#)

[ADDING REMOTE REPOSITORY](#)

[REMOTE REPOSITORIES](#)

[ADD NEW FILES](#)

[GIT COMMIT](#)

[GIT PUSH](#)

[FETCH REMOTE REPOSITORY](#)

[GIT BRANCH](#)

[GIT CHECKOUT](#)

[CREATING REPOSITORY](#)

[GIT STATUS](#)

[GIT DIFF](#)

[GIT REMOVE](#)

[GIT MOVE](#)

[GIT LOG](#)

Git Lab Manual

About

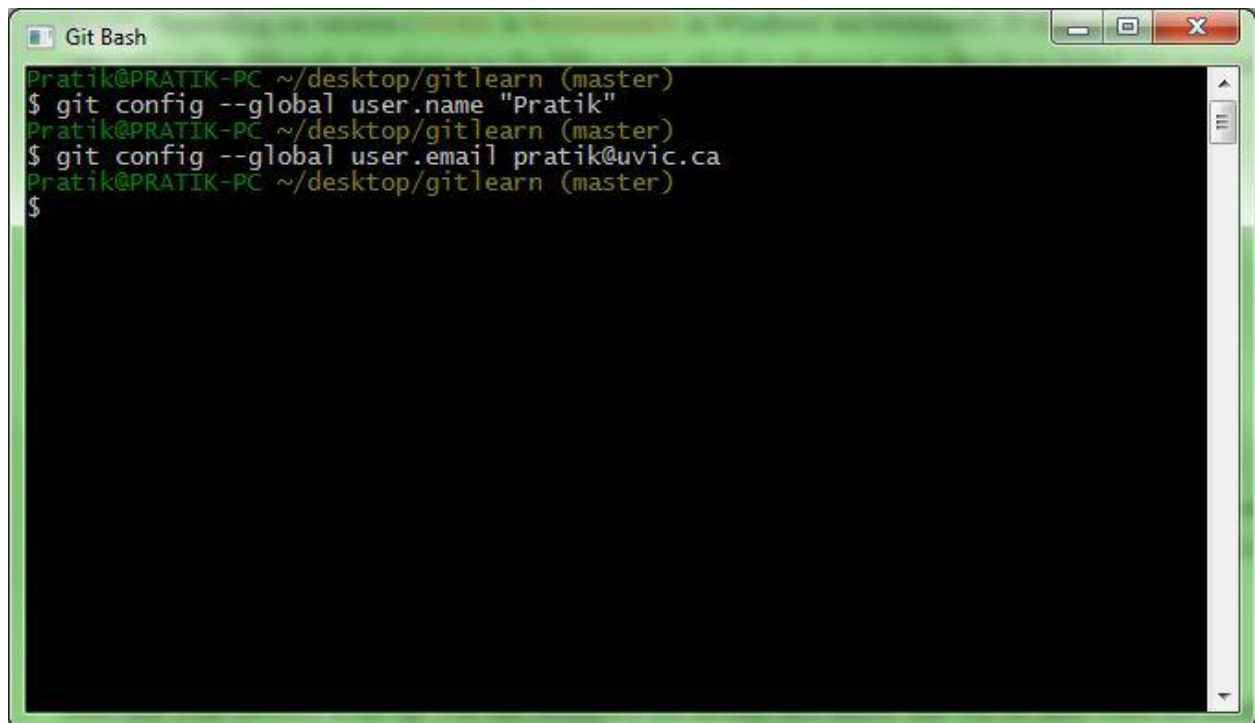
Source code management (SCM) is used to track modifications to a source code repository. SCM tracks a running history of changes to a code base and helps resolve conflicts when merging updates from multiple contributors. SCM is also synonymous with Version control.

As software projects grow in lines of code and contributor headcount, the costs of communication overhead and management complexity also grow. SCM is a critical tool to alleviate the organizational strain of increasing development costs.

STARTING WITH GIT

To start with git bash, we need to authenticate ourselves so that for every commit, git can use this information. collectively, the “Parties”) covenant and agree as follows:

```
git config --global user.name "Pratik"  
git config --global user.email pratik@uvic.ca
```

A screenshot of a Git Bash terminal window. The window title is "Git Bash". The terminal shows the following commands and output:
Pratik@PRATIK-PC ~/desktop/gitlearn (master)
\$ git config --global user.name "Pratik"
Pratik@PRATIK-PC ~/desktop/gitlearn (master)
\$ git config --global user.email pratik@uvic.ca
Pratik@PRATIK-PC ~/desktop/gitlearn (master)
\$
The terminal has a green border and a scrollbar on the right side.

We can remove -- global option to override this information for some other projects.

GIT CONFIG

It will show all the attributes of the configuration file which we have set.

```
git config --list
```

```
Git Bash
Pratik@PRATIK-PC ~/desktop/gitlearn (master)
$ git config --list
core.symlinks=false
core.autocrlf=true
color.diff=auto
color.status=auto
color.branch=auto
color.interactive=true
pack.packsizelimit=2g
help.format=html
http.sslcainfo=/bin/curl-ca-bundle.crt
sendemail.smtpserver=/bin/msmtp.exe
diff.astextplain.textconv=astextplain
rebase.autosquash=true
gui.recentrepo=C:/Users/Pratik/Desktop/Git Learn
user.name=Pratik
user.email=pratik@uvic.ca
core.repositoryformatversion=0
core.filemode=false
core.bare=false
core.logallrefupdates=true
core.symlinks=false
core.ignorecase=true
core.hidedotfiles=dotGitOnly
gui.wmstate=normal
gui.geometry=887x427+366+51 171 192
```

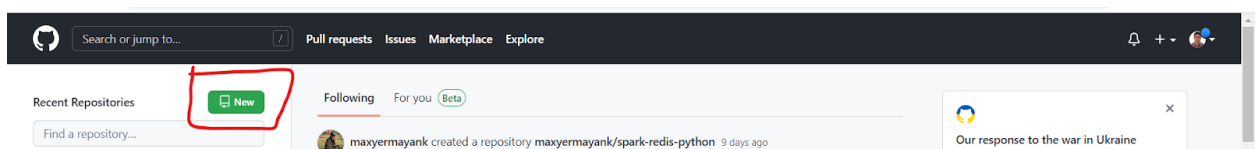
GIT HELP

It will open a browser window as help for the config command.

```
git help config
```

ADDING REMOTE REPOSITORY

1. Click on the URL ([Sign in to GitHub](#)) to open the GitHub login page
2. Enter your GitHub credentials and hit **sign in**.
3. Click on **New** button to create a new repository




- Put your repository name in the **Repository Name field**. Keep your repository public.

Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository.](#)

Owner *

 santoshnist2011 ▾

Repository name *

/

Great repository names are short and memorable. Need inspiration? How about [turbo-adventure?](#)

Description (optional)

☒  **Public**

Anyone on the internet can see this repository. You choose who can commit.

☐  **Private**

You choose who can see and commit to this repository.

Initialize this repository with:

Skip this step if you're importing an existing repository.

☐ **Add a README file**

This is where you can write a long description for your project. [Learn more.](#)

Add .gitignore

Choose which files not to track from a list of templates. [Learn more.](#)

.gitignore template: **None** ▾

Choose a license

A license tells others what they can and can't do with your code. [Learn more.](#)

- Click on **Create Repository button**

 You are creating a public repository in your personal account.

Create repository

Create a new repository in Github, then add a remote repository using:

```
git remote add santosh
```

```
https://github.com/santoshnist2011/remote_Repo_demo.git
```

```

hp@DESKTOP-MBHSJRH MINGW64 ~/Desktop/gitlab/gitlab (main)
$ git remote -v
origin https://github.com/santoshnist2011/gitlab.git (fetch)
origin https://github.com/santoshnist2011/gitlab.git (push)

hp@DESKTOP-MBHSJRH MINGW64 ~/Desktop/gitlab/gitlab (main)
$ git remote add santosh https://github.com/santoshnist2011/remote_repo_demo.git

hp@DESKTOP-MBHSJRH MINGW64 ~/Desktop/gitlab/gitlab (main)
$ git remote -v
origin https://github.com/santoshnist2011/gitlab.git (fetch)
origin https://github.com/santoshnist2011/gitlab.git (push)
santosh https://github.com/santoshnist2011/remote_repo_demo.git (fetch)
santosh https://github.com/santoshnist2011/remote_repo_demo.git (push)

```

CLONING REPOSITORY

Instead of using https://, you can use git:// protocol or user@server:/path.git, which uses the SSH transfer protocol.

```
git clone https://github.com/santoshnist2011/gitlab.git
```

```

hp@DESKTOP-MBHSJRH MINGW64 ~/Desktop/gitlab/gitlab (main)
$ git clone https://github.com/santoshnist2011/gitlab.git
Cloning into 'gitlab'...
remote: Enumerating objects: 9, done.
remote: Counting objects: 100% (9/9), done.
remote: Compressing objects: 100% (6/6), done.
remote: Total 9 (delta 0), reused 3 (delta 0), pack-reused 0
Receiving objects: 100% (9/9), done.

```

REMOTE REPOSITORIES

To check all remote handles, from a particular repository. We can use the **git remote** command. Cloning a repository from any Git server gives a default name origin to that server.

```

hp@DESKTOP-MBHSJRH MINGW64 ~/Desktop/gitlab/gitlab (main)
$ git remote
origin

hp@DESKTOP-MBHSJRH MINGW64 ~/Desktop/gitlab/gitlab (main)
$ git remote -v
origin https://github.com/santoshnist2011/gitlab.git (fetch)
origin https://github.com/santoshnist2011/gitlab.git (push)

```

-v option gives the full name of the repository that git has stored with a short name.

ADD NEW FILES

Create a first file gitcommands.txt and check the status. We need to add this new gitcommands.txt file and make it ready to commit but we can see the cloned repository.

```
git add gitcommands.txt
```

```
hp@DESKTOP-MBHSJRH MINGW64 ~/Desktop/gitlab/gitlab (main)
$ git add gitcommands.txt
warning: LF will be replaced by CRLF in gitcommands.txt.
The file will have its original line endings in your working directory
```

Create a file second.txt in the project folder and check the status again.

```
hp@DESKTOP-MBHSJRH MINGW64 ~/Desktop/gitlab/gitlab (main)
$ vi testFile.txt

hp@DESKTOP-MBHSJRH MINGW64 ~/Desktop/gitlab/gitlab (main)
$ git status
On branch main
Your branch is up to date with 'origin/main'.

Untracked files:
  (use "git add <file>..." to include in what will be committed)
    gitlab/
    testFile.txt

nothing added to commit but untracked files present (use "git add" to track)
```

Add file testFile.txt to stage for commit.

```
git add testFile.txt
```

```

hp@DESKTOP-MBHSJRH MINGW64 ~/Desktop/gitlab/gitlab (main)
$ git add testFile.txt
warning: LF will be replaced by CRLF in testFile.txt.
The file will have its original line endings in your working directory

hp@DESKTOP-MBHSJRH MINGW64 ~/Desktop/gitlab/gitlab (main)
$ git status
On branch main
Your branch is up to date with 'origin/main'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    new file:   testFile.txt

Untracked files:
  (use "git add <file>..." to include in what will be committed)
    gitlab/

```

Modify the testFile.txt file and check git status again.

While checking again, you will find testFile.txt is in both staged and unstaged area.

Git stage file as you run git add command, so if you commit now git will send a testFile.txt version which you added it earlier and is in staged area.

```

hp@DESKTOP-MBHSJRH MINGW64 ~/Desktop/gitlab/gitlab (main)
$ vi testFile.txt

hp@DESKTOP-MBHSJRH MINGW64 ~/Desktop/gitlab/gitlab (main)
$ git status
On branch main
Your branch is up to date with 'origin/main'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    new file:   testFile.txt

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
    modified:   testFile.txt

Untracked files:
  (use "git add <file>..." to include in what will be committed)
    gitlab/

```

After each modification, you need to add a file.

GIT COMMIT

Check if all the files are staged, then you can commit. If any file is in the unstaged area that won't go in commit.

Just type git commit to commit. Git commit command gives you access to write some message before committing.

```
git commit
```

```
# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
#
# On branch main
# Your branch is ahead of 'origin/main' by 1 commit.
#   (use "git push" to publish your local commits)
#
# Changes to be committed:
#   modified:   testFile.txt
#
# Untracked files:
#   gitlab/
#
```

```
hp@DESKTOP-MBHSJRH MINGW64 ~/Desktop/gitlab/gitlab (main)
$ git commit
Aborting commit due to empty commit message.

hp@DESKTOP-MBHSJRH MINGW64 ~/Desktop/gitlab/gitlab (main)
$ git commit
[main 2282b35] Insert
1 file changed, 1 insertion(+)

hp@DESKTOP-MBHSJRH MINGW64 ~/Desktop/gitlab/gitlab (main)
$ git commit
On branch main
Your branch is ahead of 'origin/main' by 2 commits.
  (use "git push" to publish your local commits)

Untracked files:
  (use "git add <file>..." to include in what will be committed)
  gitlab/

nothing added to commit but untracked files present (use "git add" to track)
```

This will commit all your changes which are staged with SHA1 reference.
Without Message in editor

```
git commit -m "added new lines in the file"
```

If you modify some file, for example – testFile.txt and try to commit without adding it. Then it won't be possible. First, you have to add and then commit. You can use commit with option –a which will commit and add all the unstaged files.

```
hp@DESKTOP-MBHSJRH MINGW64 ~/Desktop/gitlab/gitlab (main)
$ git commit -m "added new lines in the file"
[main 9ec70f7] added new lines in the file
1 file changed, 4 insertions(+)
create mode 100644 testFile.txt
```

We can see from our commit, which branch you committed to (main), what SHA-1 checksum the commit has (9ec70f7), how many files were changed, and statistics about lines added and removed in the commit. Every time you are committing, you are taking snapshots of your staging area.

GIT PUSH

To share your work with others you can push your branch to the origin or to some other remote repository.

git push [remote name][branch name]

```
hp@DESKTOP-MBHSJRH MINGW64 ~/Desktop/gitlab/gitlab (testing)
$ git push sant testing
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 4 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 335 bytes | 167.00 KiB/s, done.
Total 3 (delta 1), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To https://github.com/santoshnist2011/remote_repo_demo.git
b74b2b8..953eb09 testing -> testing
```

FETCH REMOTE REPOSITORY

We can fetch from a short named remote repository sant using:

git fetch [shortname]

```
git fetch sant
```

```
hp@DESKTOP-MBHSJRH MINGW64 ~/Desktop/gitlab/gitlab (main)
$ git fetch sant
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), 605 bytes | 18.00 KiB/s, done.
From https://github.com/santoshnist2011/remote_Repo_demo
* [new branch]      main       -> sant/main
```

After fetching we will have all the references to all the branches from this remote repository. Fetch only pulls data to your local repository, any work you want to merge you have to do it manually.

git fetch origin will fetch any new data pushed to the server you cloned.

Because origin is the default name of that server you cloned from.

GIT BRANCH

Create a git branch named testing.

```
git branch testing
```

```
hp@DESKTOP-MBHSJRH MINGW64 ~/Desktop/gitlab/gitlab (main)
$ git branch testing
```

GIT CHECKOUT

Switch to the branch named testing.

```
git checkout testing
```

```
hp@DESKTOP-MBHSJRH MINGW64 ~/Desktop/gitlab/gitlab (main)
$ git checkout testing
Switched to branch 'testing'
A       gitMV_Command_Test.txt
D       testFile.txt
```

We can combine the previous two commands to create a git branch and then switch it to using the `-b` option with `checkout`.

`git checkout -b testing`


Now head will point to testing.

Now you started working on your branch, made a few changes, and commit.

CREATING REPOSITORY

It will create a `.git` directory in your project directory.

```
git init
```

 MINGW64:/c/Users/hp/Desktop/gitlab

```
hp@DESKTOP-MBHSJRH MINGW64 ~/Desktop/gitlab
$ git init
Initialized empty Git repository in C:/Users/hp/Desktop/gitlab/.git/
hp@DESKTOP-MBHSJRH MINGW64 ~/Desktop/gitlab (master)
$ |
```

GIT STATUS

If you run this command, just after cloning a repository. You will see there is nothing to commit all files are tracked and unmodified.

```
git status
```

```
hp@DESKTOP-MBHSJRH MINGW64 ~/Desktop/gitlab/gitlab (main)
$ git status
On branch main
Your branch is up to date with 'origin/main'.

Untracked files:
  (use "git add <file>..." to include in what will be committed)
    gitcommands.txt

nothing added to commit but untracked files present (use "git add" to track)
```

You can use the `-s` option for checking status in a short and concise way. It shows the result which has the below terminology.

? Untracked files

A Added files

M Modified files

R Rename

GIT DIFF

You can check your staged and unstaged changes through git status but git diff will give details of what is changed in files.

To see what you have changed but not yet staged, type git diff with no other arguments.

```
git diff --staged
```

```
hp@DESKTOP-MBHSJRH MINGW64 ~/Desktop/gitlab/gitlab (main)
$ git diff --staged
diff --git a/testFile.txt b/testFile.txt
new file mode 100644
index 0000000..b834c3f
--- /dev/null
+++ b/testFile.txt
@@ -0,0 +1 @@
+I am creating this file for testing the second file insert in the cloned repo.
```

It gives the details of files that you have staged and will go into the next commit.

It's important to note that git diff by itself doesn't show all changes made since your last commit — only changes that are still unstaged. If you have staged all your changes, git diff will give you no output.

```
np@DESKTOP-MBHSJRH MINGW64 ~/Desktop/gitlab/gitlab (main)
$ git diff --staged
diff --git a/testFile.txt b/testFile.txt
new file mode 100644
index 0000000..16cb5cd
--- /dev/null
+++ b/testFile.txt
@@ -0,0 +1,4 @@
+I am creating this file for testing the second file insert in the cloned repo.
+Second edit
+
+Use of Diff command in git
```

GIT REMOVE

To remove files that are tracked or in the staging area, you need the git rm command and then commit.

Removes all the files that end with ~

```
git rm \*~
```

If you simply remove the file from your working directory using the rm command, the file will be in the unstaged area("Changes not staged for commit").

GIT MOVE

Git explicitly does not track file movement, and it does not store any metadata like if you rename the file. But git has mv command.

```

hp@DESKTOP-MBHSJRH MINGW64 ~/Desktop/gitlab/gitlab (main)
$ git mv testFile.txt gitMV_Command_Test.txt

hp@DESKTOP-MBHSJRH MINGW64 ~/Desktop/gitlab/gitlab (main)
$ git status
On branch main
Your branch is ahead of 'origin/main' by 2 commits.
  (use "git push" to publish your local commits)

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        renamed:    testFile.txt -> gitMV_Command_Test.txt

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        gitlab/

```

This git mv command is actually equivalent to the three commands stated below example.

```

hp@DESKTOP-MBHSJRH MINGW64 ~/Desktop/gitlab/gitlab (main)
$ ls
README.md  gitMV_Command_Test.txt  gitcommands.txt  gitlab/

hp@DESKTOP-MBHSJRH MINGW64 ~/Desktop/gitlab/gitlab (main)
$ git rm gitMV_Command_Test.txt
error: the following file has changes staged in the index:
        gitMV_Command_Test.txt
(use --cached to keep the file, or -f to force removal)

hp@DESKTOP-MBHSJRH MINGW64 ~/Desktop/gitlab/gitlab (main)
$ git add gitMV_Command_Test.txt

hp@DESKTOP-MBHSJRH MINGW64 ~/Desktop/gitlab/gitlab (main)
$ git status
On branch main
Your branch is ahead of 'origin/main' by 2 commits.
  (use "git push" to publish your local commits)

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        renamed:    testFile.txt -> gitMV_Command_Test.txt

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        gitlab/

```

GIT LOG

Gives you details with checksum and messages in chronological order of commits.

git log

Option -p will show differences introduced in each commit and -2 will restrict it to only last two entries.

git log -p -2

If we want to see some abbreviated stats for each commit. It also puts summary of information at the end.

git log --stat

Option --pretty changes the log output to formats other than the default. There are some predefined formats.

git log --pretty=format:"%h - %an, %ar : %s"

Where **%h** is abbr. commit hash, **%an** is author name, **%ar** is author date(relative), and **%s** is subject or message.

%ad can also be used for date format.